الجمهورية الجزائرية الديمقراطية الشعبية
People's Democratic and Republic of Algeria

Ministry of Higher Education and Scientific Research
National School of Cyber Security

NSCS
المدرسة الوطنية العليا في الأمن السيبراني
NATIONAL SCHOOL OF CYBERSECURITY

وزارة التعليم العالي والبحث العلمي
المدرسة الوطنية العليا في الأمن السيبراني

Tutorials 2: linear linked lists

February 2025

Objectives: The primary goal of this series of exercises is to develop a deep understanding of **linear linked lists** and their applications in various computing contexts. The exercises are designed to progressively build knowledge, starting with basic linked list operations and advancing to more complex use cases.

1. **Create a Linked List from Input Data**: give a module to construct a linear linked list from a set of user-provided data.
2. **Length of a Linked List**: Write a module to calculate the length of a linked list (number of nodes).
3. **Find Element with the Most Occurrences**: Write a module to search for the element that occurs the most in the linked list.
4. **Access Element by Value**: Write a module that returns the first node containing a given value in the linked list.
5. **Access Element by Position**: Write a module to access the node at a specific position in the linked list.
6. **Delete by Value**: Create a module to delete the first node that contains a specified value in the linked list.
7. **Delete by Position**: Write a module to delete a node at a specified position in the linked list.
8. **Insert by Position**: Write a module to insert a new node at a specific position in the linked list.
9. **Merge Two Sorted Lists**: Write a module that merges two sorted linked lists into one sorted list.
10. **Split a List Based on a Criterion**: Create a module to split a linked list into two based on a specific criterion (e.g., even/odd).
11. **Find the Node with the Minimum Value**: Implement a module to find the node with the smallest value in the linked list.
12. **Merge Two Linked Lists Using No Additional Memory**: Write a module that merges two linked lists into one without allocating additional memory.
13. **Delete All Nodes with a Specific Value**: Write a module to delete all nodes with a given value from the linked list.
14. **Reverse the Linked List**: Implement a module that reverses the linked list
15. **Check if a Linked List is a Palindrome**: Implement a function to check if a linked list is a palindrome (reads the same forward and backward).
16. **Remove Duplicates from an Unsorted Linked List**: Write a module to remove duplicate values from an unsorted linked list without using extra memory.

17. **Add a Node to the End Only If the List is Empty**: Implement a module that adds a node to the end of the linked list only if the list is empty.
18. **Sort a Linked List Using Selection Sort**: Implement a selection sort algorithm to sort the elements of a linked list.
19. **Reverse a Linked List in Groups of Size K**: Write a module to reverse the linked list in groups of size K.
20. **Rotate a Linked List**: Write a function to rotate the linked list (move nodes to the left or right in a circular fashion).
21. **Delete Nodes with Values Below a Threshold**: Implement a function to delete all nodes from a linked list whose values are below a given threshold.
22. **Create a Doubly Linked List from Input Data**: Implement a module to create a doubly linked list from user-provided data.
23. **Insert an Element in a Doubly Linked List**: Write a function to insert an element into a doubly linked list.
24. **Delete an Element from a Doubly Linked List**: Create a function to delete an element from a doubly linked list.
25. **Construct a Circular Linked List**: Implement a module to create a circular linked list where the last node points back to the first node.
26. **Bidirectional Linked List**: Build a bidirectional (doubly) linked list and implement modules to insert and delete nodes at any position.
27. **Implement a Circular Doubly Linked List**: Create a circular doubly linked list where nodes point to both the next and previous nodes, and the list wraps around itself.